ESTRUCTURAS DE DATOS

Curso 2018/19

PRÁCTICA 3

Mapas

Instrucciones

- *Se debe completar en una sesión.*
- *Práctica individual.*
- *Lee el enunciado completo antes de comenzar. Los comentarios incluidos en el código también pueden proporcionar información útil.*
- *Se habilitará una tarea para que entregues el código desarrollado.*
- *La práctica será APTA si se superan todos los test de validación proporcionados.*

Un banco posee una serie de cuentas en la que los clientes almacenan una cantidad de dinero. Una de las operaciones más habituales es la transferencia de dinero entre cuentas del banco. En una transferencia existe una cuenta de origen que transfiere una cantidad a una cuenta destino. Al realizar la operación, el saldo de la cuenta de origen se reduce en la cantidad transferida, mientras que el saldo de la cuenta de destino se incrementa en la misma cantidad.

Deseamos escribir una clase **`Banco`** que guarde el saldo de las cuentas existentes y un histórico de las transferencias producidas entre ellas. Una cuenta tiene un código, **`String`** y un saldo, **`int`**. No puede haber cuentas con códigos repetidos. Para representar una transacción entre cuentas definimos la clase **`Transferencia`**.

La clase Banco almacenará los datos en dos mapas:

- **`HashMap<String, Integer> cuentas`**: por cada código de cuenta guarda su saldo.
- **`HashMap<String, List<Transferencia>> desglose`**: por cada código de cuenta guarda un listado con las transferencias en las que se ha visto implicada, ya sea como origen o como destino.

Ejercicio 1

Escribe el constructor de la clase **`Banco`**. Tomará dos listas, una con códigos de cuenta y otra con los saldos respectivos, y guardará los datos correspondientes. Si aparecen códigos repetidos, se almacenará solo uno de ellos con la suma de todos los saldos. Puede haber cuentas con saldos negativos.

```
public Banco(List<String> codigos, List<Integer> saldos)
```

En caso de que alguno de los parámetros sea **`null`**, las listas sean de distinto tamaño o el código o saldo alguna de alguna de las cuentas sea **`null`** lanzará una excepción **`IllegalArgumentException`**.

Ejercicio 2

Escribe un método que devuelva el saldo de una cuenta.

```
public int consulta(String codigo)
```

Lanzará una excepción **`IllegalArgumentException`** `si` el código de la cuenta no existe.

Ejercicio 3

Añade a la clase un método que procese una transferencia, esto es, que modifique los saldos de la cuenta origen y destino según la cantidad transferida y almacena la transferencia para futuras consultas. La cantidad podrá ser negativa indicando que la transferencia se realiza desde la cuenta destino a la de origen. El método no realizará ni almacenará la transferencia en el caso de que la cuenta origen no disponga de suficiente saldo positivo para realizar la transferencia. Devolverá **`false`** en este último caso y **`true`** si se realiza la transferencia con éxito.

```
public boolean asiento(Transferencia tr)
```

El método lanza una excepción **`IllegalArgumentException`** en el caso de que alguna de las cuentas no exista.

Ejercicio 4

Escribe un método **`historico`** que devuelva una lista con todas las transferencias en cualquier sentido entre dos cuentas, primera y segunda. Al devolver el resultado, en las transferencias aparecerá siempre **`primera`** como cuenta de origen y **`segunda`** como cuenta destino, teniendo la cantidad un valor negativo si fuera necesario para indicar la dirección de la transferencia. Devolverá una lista vacía si se trata de cuentas con el mismo código.

```
public List<Transferencia> historico(String primera, String segunda)
```

El método lanzará una excepción **`IllegalArgumentException`** en el caso de que alguno de los códigos de cuenta sean **`null`** o no existan.